ncem

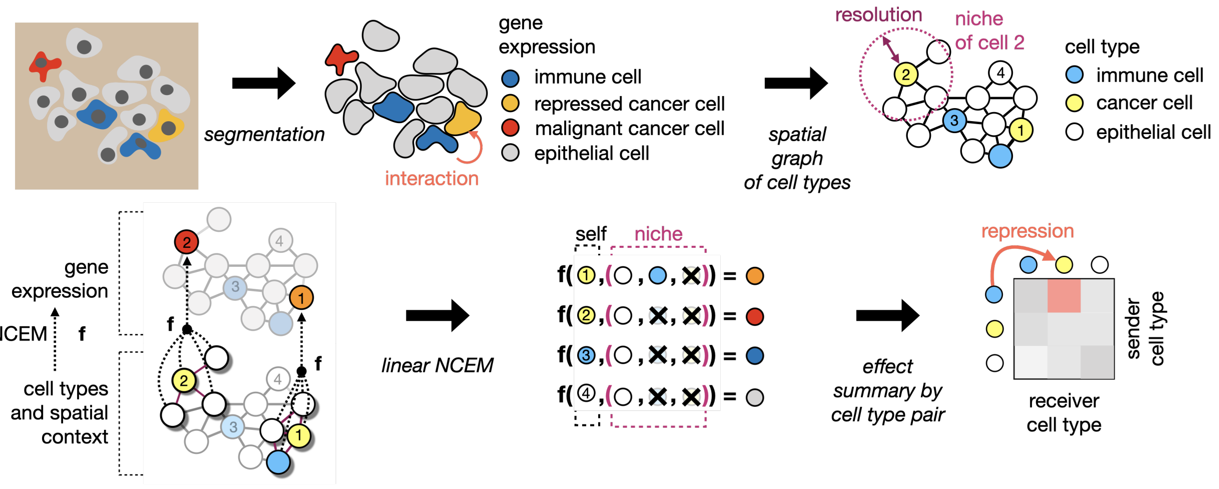
Release 0.1.5

David S. Fischer, Anna C. Schaar

Jun 27, 2023

GENERAL

NCEM is a tool for the inference of cell-cell communication in spatial molecular data.



CHAPTER ONE

MANUSCRIPT

Please see our manuscript [Fischer *et al.*, 2023] in **Nature Biotechnology** to learn more.

CHAPTER

TWO

NCEM'S KEY APPLICATION

- Node-centric expression models (NCEMs) are proposed to improve cell communication inference by using spatial graphs of cells to constrain axes of cellular communication.
- NCEMs can be used for cell communication inference captured with different spatial profiling technologies and are not limited to receptor-ligand signaling.
- NCEMs can be applied to deconvoluted spot transcriptomics.
- Dependencies inferred by NCEMs are directional.

CHAPTER

THREE

GETTING STARTED WITH NCEM

You can install *ncem* via pip from PyPI:

```
$ pip install ncem
```

CHAPTER

FOUR

CONTRIBUTING TO NCEM

We are happy about any contributions! Before you start, check out our contributor guide.

4.1 Installation

4.1.1 Stable release

To install ncem, run this command in your terminal:

```
$ pip install ncem
```

This is the preferred method to install ncem, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

4.1.2 From sources

The sources for ncem can be downloaded from the Github repo. Please note that you require poetry to be installed.

You can either clone the public repository:

```
$ git clone git://github.com/theislab/ncem
```

Or download the tarball:

```
$ curl -OJL https://github.com/theislab/ncem/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ make install
```

4.2 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the BSD license and highly welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

4.2.1 How to add a dataloader

Overview of contributing dataloaders to ncem.

1. **Install ncem.** Clone ncem into a local repository from *development* branch and install via pip.

```
cd target_directory
git clone https://github.com/theislab/ncem.git
git checkout development
# git pull  # use this to update your installation
cd ncem  # go into ncem directory
pip install -e .  # install
```

2. **Create a new dataloader in *data.py*** Your dataloader should be a new class in *data.py* (ideally named by first author, e.g. DataLoaderZhang) and should contain the following functions *_register_celldata*, *_register_img_celldata* and *_register_graph_features*.

   _register_celldata creates an AnnData object called *celldata* of the complete dataset with feature names stored in *celldata.var_names*. Cell type annotations are stored in *celldata.obs*. *celldata.uns['metadata']* should contain the naming conventions of files and columns in obs.

   _register_img_celldata then automatically splits the *celldata* into a dictionary of AnnData object with one AnnData object per image in the dataset.

   _register_graph_features can be added in case of additional graph features, e.g. disease status of images.

   Additionally, each dataloader should have a class attribute *cell_type_merge_dict* which provides a logic on how to merge cell types in any subsequent analysis. This can be helpful when considering datasets with fine cell type annotations and a coarser annotation is wanted.

3. **Make loader public (Optional).** You can contribute the data loader to public ncem as code through a pull request. Note that you can also just keep the data loader in your local installation if you do not want to make it public.

```
# make sure you are in the top-level ncem directory from step 1
git add *
git commit  # enter your commit description
# Next make sure you are up to date with dev
git checkout development
git pull
git checkout YOUR_BRANCH_NAME
git merge development
git push  # this starts the pull request.
```

In any case, feel free to open an GitHub issue on on the Issue Tracker.

4.2.2 How to report a bug

Report bugs on the Issue Tracker.

4.2.3 How to request a feature

Request features on the Issue Tracker.

4.2.4 How to set up your development environment

You need Python 3.7+ and the following tools:

- Poetry
- Nox
- nox-poetry

You can install them with:

```
$ pip install poetry nox nox-poetry
```

Install the package with development requirements:

```
$ make install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run ncem
```

4.2.5 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

4.2.6 How to submit changes

Open a pull request to submit changes to this project against the `development` branch.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains a high code coverage.
- If your changes add functionality, update the documentation accordingly.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

4.3 Ecosystem

4.3.1 squidpy

squidpy [Palla *et al.*, 2022] provides an environment of tools that can be used to analysis spatial transcriptomnics in python.

4.3.2 scanpy

scanpy [Wolf *et al.*, 2018] provides an environment of tools that can be used to analysis single-cell data in python.

4.4 Credits

4.4.1 Development Lead

- David Fischer <david.fischer@helmholtz-muenchen.de>
- Anna Schaar <anna.schaar@helmholtz-muenchen.de>

4.4.2 Contributors

None yet. Why not be the first?

4.5 References

4.6 Contributor Covenant Code of Conduct

4.6.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.6.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.6.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.6.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

4.6.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

4.6.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

4.7 API

Import ncem as:

```
import ncem
```

4.7.1 Tools

NCEM tools containing linear models, variance decomposition and ablation study.

tl.linear_ncem(adata, key_type, key_graph[, …])	Fit a linear NCEM based on an adata instance and save fits in instance.
tl.linear_ncem_deconvoluted(adata, …[, …])	Fit a linear NCEM based on deconvoluted data in an adata instance and save fits in instance.
tl.differential_ncem(adata, …[, formula, …])	Fit a differential NCEM based on an adata instance and save fits in instance.
tl.differential_ncem_deconvoluted(adata, …)	Fit a differential NCEM based on deconvoluted data in an adata instance and save fits in instance.
tl.spline_linear_ncem(adata, df, …[, …])	Fit a linear NCEM based on an adata instance and save fits in instance.
tl.spline_linear_ncem_deconvoluted(adata, …)	Fit a linear NCEM based on deconvoluted data in an adata instance and save fits in instance.
tl.spline_differential_ncem(adata, df, …)	Fit a differential NCEM based on an adata instance and save fits in instance.
tl.spline_differential_ncem_deconvoluted(…)	Fit a differential NCEM based on deconvoluted data in an adata instance and save fits in instance.

ncem.tl.linear_ncem

`ncem.tl.`**`linear_ncem`**(*adata: anndata._core.anndata.AnnData*, *key_type: str*, *key_graph: str*, *formula: str = '~0'*, *type_specific_confounders: List[str] = []*)

Fit a linear NCEM based on an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. formula: Description of batch covariates as linear model. Do not include intercept, cell type, or niche as

this is automatically added.

key_type: Key of type annotation in .obs. key_graph: Key of spatial neighborhood graph in .obsp. type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell

types, ie confounders that act on the cell type level. Global confounders can be added in the formula.

Returns:

ncem.tl.linear_ncem_deconvoluted

`ncem.tl.`**`linear_ncem_deconvoluted`**(*adata: anndata._core.anndata.AnnData*, *key_deconvolution: str*, *formula: str = '~0'*, *type_specific_confounders: List[str] = []*)

Fit a linear NCEM based on deconvoluted data in an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. Note on placement of deconvolution output:

- type abundances must in be in .obsm[key_deconvolution] with cell type names as columns
- spot- and type-specific gene expression results must be layers named after types

formula: Description of batch covariates as linear model. Do not include intercept, cell type, or niche as
this is automatically added.

key_deconvolution: Key of type deconvolution in .obsm. type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell

types, ie confounders that act on the cell type level. As the formula is used for each index cell, this is equivalent to adding these terms into the formula.

Returns:

ncem.tl.differential_ncem

`ncem.tl.`**`differential_ncem`**(*adata: anndata._core.anndata.AnnData*, *key_differential: str*, *key_graph: str*, *key_type: str*, *formula: str = '~0'*, *type_specific_confounders: List[str] = []*)

Fit a differential NCEM based on an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. formula: Description of batch covariates as linear model. Do not include intercept, cell type, niche, or the

differential term as this is automatically added.

key_differential: Key of condition annotation in .obs. This will be used for testing. key_graph: Key of spatial neighborhood graph in .obsp. key_type: Key of type annotation in .obs. type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell

types, ie confounders that act on the cell type level. Global confounders can be added in the formula.

Returns:

ncem.tl.differential_ncem_deconvoluted

`ncem.tl.differential_ncem_deconvoluted`(*adata: anndata._core.anndata.AnnData*, *key_differential: str*, *key_deconvolution: str*, *formula: str = '~0'*, *type_specific_confounders: List[str] = []*)

Fit a differential NCEM based on deconvoluted data in an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. Note on placement of deconvolution output:

- type abundances must in be in .obsm[key_deconvolution] with cell type names as columns
- spot- and type-specific gene expression results must be layers named after types

formula: Description of batch covariates as linear model. Do not include intercept, cell type, niche, or the differential term as this is automatically added.

key_deconvolution: Key of type deconvolution in .obsm. key_differential: Key of condition annotation in .obs. This will be used for testing. type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell

types, ie confounders that act on the cell type level. As the formula is used for each index cell, this is equivalent to adding these terms into the formula.

Returns:

ncem.tl.spline_linear_ncem

`ncem.tl.spline_linear_ncem`(*adata: anndata._core.anndata.AnnData*, *df: int*, *key_1d_coord: str*, *key_graph: str*, *key_type: str*, *formula: str = '~0'*, *spline_family: str = 'cr'*, *type_specific_confounders: List[str] = []*)

Fit a linear NCEM based on an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. df: Degrees of freedom of the spline model, i.e. the number of spline basis vectors. formula: Description of batch covariates as linear model. Do not include intercept, cell type, or niche as

this is automatically added.

key_1d_coord: Key of numeric 1D coordinate of each observation in .obs. This will be used to build the spline. key_type: Key of type annotation in .obs. key_graph: Key of spatial neighborhood graph in .obsp. spline_family: The type of sline basis to use, refer also to:

https://patsy.readthedocs.io/en/latest/spline-regression.html

- “bs”: B-splines
- “cr”: natural cubic splines

- “cc”: natural cyclic splines

type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell
types, ie confounders that act on the cell type level. Global confounders can be added in the formula.

Returns:

ncem.tl.spline_linear_ncem_deconvoluted

ncem.tl.**spline_linear_ncem_deconvoluted**(*adata: anndata._core.anndata.AnnData*, *df: int*, *key_1d_coord: str*, *key_deconvolution: str*, *formula: str = '~0'*, *spline_family: str = 'cr'*, *type_specific_confounders: List[str] = []*)

Fit a linear NCEM based on deconvoluted data in an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. Note on placement of deconvolution output:

- type abundances must in be in .obsm[key_deconvolution] with cell type names as columns
- spot- and type-specific gene expression results must be layers named after types

df: Degrees of freedom of the spline model, i.e. the number of spline basis vectors. formula: Description of batch covariates as linear model. Do not include intercept, cell type, or niche as

this is automatically added.

key_1d_coord: Key of numeric 1D coordinate of each observation in .obs. This will be used to build the spline. key_deconvolution: Key of type deconvolution in .obsm. spline_family: The type of sline basis to use, refer also to:

https://patsy.readthedocs.io/en/latest/spline-regression.html

- “bs”: B-splines
- “cr”: natural cubic splines
- “cc”: natural cyclic splines

type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell
types, ie confounders that act on the cell type level. As the formula is used for each index cell, this is equivalent to adding these terms into the formula.

Returns:

ncem.tl.spline_differential_ncem

ncem.tl.**spline_differential_ncem**(*adata: anndata._core.anndata.AnnData*, *df: int*, *key_1d_coord: str*, *key_differential: str*, *key_graph: str*, *key_type: str*, *formula: str = '~0'*, *spline_family: str = 'cr'*, *type_specific_confounders: List[str] = []*)

Fit a differential NCEM based on an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. df: Degrees of freedom of the spline model, i.e. the number of spline basis vectors. formula: Description of batch covariates as linear model. Do not include intercept, cell type, niche, or the

differential term as this is automatically added.

key_1d_coord: Key of numeric 1D coordinate of each observation in .obs. This will be used to build the spline. key_differential: Key of condition annotation in .obs. This will be used for testing. key_graph: Key of spatial neighborhood graph in .obsp. key_type: Key of type annotation in .obs. spline_family: The type of sline basis to use, refer also to:

https://patsy.readthedocs.io/en/latest/spline-regression.html

- "bs": B-splines
- "cr": natural cubic splines
- "cc": natural cyclic splines

type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell types, ie confounders that act on the cell type level. Global confounders can be added in the formula.

Returns:

ncem.tl.spline_differential_ncem_deconvoluted

`ncem.tl.`**`spline_differential_ncem_deconvoluted`**(*adata: anndata._core.anndata.AnnData*, *df: int*, *key_1d_coord: str*, *key_differential: str*, *key_deconvolution: str*, *formula: str = '~0'*, *spline_family: str = 'cr'*, *type_specific_confounders: List[str] = []*)

Fit a differential NCEM based on deconvoluted data in an adata instance and save fits in instance.

Saves fits and Wald test output into instance.

Args: adata: AnnData instance with data and annotation. Note on placement of deconvolution output:

- type abundances must in be in .obsm[key_deconvolution] with cell type names as columns
- spot- and type-specific gene expression results must be layers named after types

df: Degrees of freedom of the spline model, i.e. the number of spline basis vectors. formula: Description of batch covariates as linear model. Do not include intercept, cell type, niche, or the

differential term as this is automatically added.

key_1d_coord: Key of numeric 1D coordinate of each observation in .obs. This will be used to build the spline. key_deconvolution: Key of type deconvolution in .obsm. key_differential: Key of condition annotation in .obs. This will be used for testing. spline_family: The type of sline basis to use, refer also to:

https://patsy.readthedocs.io/en/latest/spline-regression.html

- "bs": B-splines
- "cr": natural cubic splines
- "cc": natural cyclic splines

type_specific_confounders: List of confounding terms in .obs to be added with an interaction term to cell types, ie confounders that act on the cell type level. As the formula is used for each index cell, this is equivalent to adding these terms into the formula.

Returns:

4.7.2 Plotting

NCEM tools containing plotting functions.

`pl.cluster_freq`(adata, cluster_key[, title, . . .])	Plot cluster frequencies.
`pl.noise_structure`(adata, cluster_key[, . . .])	Plot cluster frequencies.
`pl.circular`(adata, alpha, scale_edge[, . . .])	Plot cluster frequencies.
`pl.circular_rotated_labels`(adata, alpha, . . .)	Plot cluster frequencies.
`pl.ablation`(adata[, figsize])	Plot of ablation study results

ncem.pl.cluster_freq

`ncem.pl.`**`cluster_freq`**(*adata: anndata._core.anndata.AnnData, cluster_key: str, title: Optional[str] = None, figsize: Optional[Tuple[float, float]] = None, dpi: Optional[int] = None, save: Optional[Union[str, pathlib.Path]] = None, ax: Optional[matplotlib.axes._axes.Axes] = None*) → None

Plot cluster frequencies.

Args: adata: AnnData instance with data and annotation. cluster_key: title: figsize: dpi: save: ax:

ncem.pl.noise_structure

`ncem.pl.`**`noise_structure`**(*adata: anndata._core.anndata.AnnData, cluster_key: str, title: Optional[str] = None, figsize: Optional[Tuple[float, float]] = None, dpi: Optional[int] = None*) → None

Plot cluster frequencies.

Args: adata: AnnData instance with data and annotation. cluster_key: title: figsize: dpi:

ncem.pl.circular

`ncem.pl.`**`circular`**(*adata, alpha, scale_edge, pvals_key: str = 'ncem_fdr_pvals', params_key: str = 'ncem_params', figsize=(10, 5), edge_type: str = 'magnitude', clip_edges: int = 0*)

Plot cluster frequencies.

Args: adata: AnnData instance with data and annotation. alpha: scale_edge: params_key: pvals_key: figsize: edge_type: clip_edges:

ncem.pl.circular_rotated_labels

`ncem.pl.`**`circular_rotated_labels`**(*adata, alpha, scale_edge, pvals_key: str = 'ncem_fdr_pvals', params_key: str = 'ncem_params', figsize=(10, 5), edge_type: str = 'magnitude', clip_edges: int = 0, text_space: float = 1.15*)

Plot cluster frequencies.

Args: adata: AnnData instance with data and annotation. alpha: scale_edge: params_key: pvals_key: figsize: edge_type: clip_edges: text_space:

ncem.pl.ablation

ncem.pl.**ablation**(*adata: anndata._core.anndata.AnnData*, *figsize: Tuple[float, float] = (3.5, 4.0)*)

Plot of ablation study results

Args: adata: AnnData instance with fits saved. figsize:

Returns: Plot

4.7.3 Model classes: *models*

Model classes from ncem for advanced use.

Classes that wrap tensorflow models.

`models.ModelCVAE`(input_shapes, latent_dim, . . .)	Model class for conditional variational autoencoder.
`models.ModelCVAEncem`(input_shapes, . . .)	Model class for NCEM conditional variational autoencoder with graph layer IND (MAX) or GCN.
`models.ModelED`(input_shapes, latent_dim, . . .)	Model class for non-spatial encoder-decoder.
`models.ModelEDncem`(input_shapes, latent_dim, . . .)	Model class for NCEM encoder-decoder with graph layer IND (MAX) or GCN.
`models.ModelInteractions`(input_shapes, . . .)	Model class for interaction model, baseline and spatial model.
`models.ModelLinear`(input_shapes, l2_coef, . . .)	Model class for linear model, baseline and spatial model.

ncem.models.ModelCVAE

class ncem.models.**ModelCVAE**(*input_shapes*, *latent_dim: int = 10*, *intermediate_dim_enc: int = 128*, *intermediate_dim_dec: int = 128*, *depth_enc: int = 1*, *depth_dec: int = 1*, *dropout_rate: float = 0.1*, *l2_coef: float = 0.0*, *l1_coef: float = 0.0*, *use_domain: bool = False*, *use_type_cond: bool = True*, *use_batch_norm: bool = False*, *scale_node_size: bool = False*, *transform_input: bool = False*, *output_layer='gaussian'*, ***kwargs*)

Model class for conditional variational autoencoder.

Methods

ncem.models.ModelCVAEncem

`class ncem.models.ModelCVAEncem`(*input_shapes*, *latent_dim: int = 10*, *intermediate_dim_enc: int = 128*, *intermediate_dim_dec: int = 128*, *depth_enc: int = 1*, *depth_dec: int = 1*, *dropout_rate: float = 0.1*, *l2_coef: float = 0.0*, *l1_coef: float = 0.0*, *cond_type: str = 'gcn'*, *cond_depth: int = 1*, *cond_dim: int = 8*, *cond_dropout_rate: float = 0.1*, *cond_activation: Union[str, keras.engine.base_layer.Layer] = 'relu'*, *cond_l2_reg: float = 0.0*, *cond_use_bias: bool = True*, *use_domain: bool = False*, *scale_node_size: bool = False*, *use_type_cond: bool = True*, *use_batch_norm: bool = False*, *transform_input: bool = False*, *output_layer: str = 'gaussian'*, ***kwargs*)

Model class for NCEM conditional variational autoencoder with graph layer IND (MAX) or GCN.

Methods

—

ncem.models.ModelED

`class ncem.models.ModelED`(*input_shapes*, *latent_dim: int = 10*, *dropout_rate: float = 0.1*, *l2_coef: float = 0.0*, *l1_coef: float = 0.0*, *enc_intermediate_dim: int = 128*, *enc_depth: int = 2*, *dec_intermediate_dim: int = 128*, *dec_depth: int = 2*, *use_domain: bool = False*, *use_type_cond: bool = True*, *scale_node_size: bool = False*, *output_layer: str = 'gaussian'*, ***kwargs*)

Model class for non-spatial encoder-decoder.

Methods

—

ncem.models.ModelEDncem

`class ncem.models.ModelEDncem`(*input_shapes*, *latent_dim: int = 10*, *dropout_rate: float = 0.1*, *l2_coef: float = 0.0*, *l1_coef: float = 0.0*, *enc_intermediate_dim: int = 128*, *enc_depth: int = 2*, *dec_intermediate_dim: int = 128*, *dec_depth: int = 2*, *cond_type: str = 'gcn'*, *cond_depth: int = 1*, *cond_dim: int = 8*, *cond_dropout_rate: float = 0.1*, *cond_activation: Union[str, keras.engine.base_layer.Layer] = 'relu'*, *cond_l2_reg: float = 0.0*, *cond_use_bias: bool = True*, *use_domain: bool = False*, *use_type_cond: bool = False*, *scale_node_size: bool = False*, *output_layer: str = 'gaussian'*, ***kwargs*)

Model class for NCEM encoder-decoder with graph layer IND (MAX) or GCN.

Methods

---

ncem.models.ModelInteractions

class ncem.models.**ModelInteractions**(*input_shapes*, *l2_coef: Optional[float] = 0.0*, *l1_coef: Optional[float] = 0.0*, *use_interactions: bool = False*, *use_domain: bool = False*, *scale_node_size: bool = False*, *output_layer: str = 'linear'*, ***kwargs*)

Model class for interaction model, baseline and spatial model.

Methods

---

ncem.models.ModelLinear

class ncem.models.**ModelLinear**(*input_shapes*, *l2_coef: float = 0.0*, *l1_coef: float = 0.0*, *use_source_type: bool = False*, *use_domain: bool = False*, *scale_node_size: bool = False*, *output_layer: str = 'linear'*, ***kwargs*)

Model class for linear model, baseline and spatial model.

Attributes: args (dict): training_model:

Raises: ValueError: If *output_layer* is not recognized.

Methods

4.8 Tutorials

We provide tutorials in separate repository.

- A tutorial for fitting and evaluating a interactions model on the MERFISH - brain dataset (interactions).

If you would like to add more tutorials, feel free to contibute or open an issue.

BIBLIOGRAPHY

[FST23] David S. Fischer, Anna C. Schaar, and Fabian J. Theis. Modeling intercellular communication in tissues using spatial graphs of cells. *Nature Biotechnology*, 41(3):332–336, March 2023. URL: https://doi.org/10.1038/s41587-022-01467-z, doi:10.1038/s41587-022-01467-z.

[PSK+22] Giovanni Palla, Hannah Spitzer, Michal Klein, David Fischer, Anna Christina Schaar, Louis Benedikt Kuemmerle, Sergei Rybakov, Ignacio L. Ibarra, Olle Holmberg, Isaac Virshup, Mohammad Lotfollahi, Sabrina Richter, and Fabian J. Theis. Squidpy: a scalable framework for spatial omics analysis. *Nature Methods*, 19(2):171–178, Feb 2022. doi:10.1038/s41592-021-01358-2.

[WAT18] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, February 2018. URL: https://doi.org/10.1186/s13059-017-1382-0, doi:10.1186/s13059-017-1382-0.

PYTHON MODULE INDEX

INDEX